SPECIFICATION

PARALLEL PROCESSING DEVICE FOR IMAGE DATA WITH SIMD ALU

TECHNICAL FIELD

The present invention relates to a data processor and, more particularly, a data processor for efficiently processing a large amount of data in a process of motion estimation or motion compensation used in video signal compressing and decompressing processes at high speed by using a processor.

BACKGROUND ART

In image or sound decompressing/compressing processes or the like, the same arithmetic process has to be repeatedly performed on a large amount of data at high speed. There is a known data processor taking the form of an SIMD (Single Instruction Multiple Data) arithmetic and logic unit (ALU) having an arithmetic and logic unit dedicated to perform the same arithmetic process, and a plurality of processor elements (arithmetic and logic units) arranged in parallel to operate the arithmetic and logic unit at high speed and operated by the same program. The SIMD ALU is described in literature "Interface", March Issue, 1998, pp. 111 to 113. Concretely, MMX technology of Pentium processor of Intel Corporation, U.S.A is known.

In an ALU of the SIMD system, to increase an operating ratio of the ALU by constantly supplying data

from a memory is an important factor which determines the performance. In a data processor in which a conventionally known central processor unit (abbreviated as CPU) and the SIMD ALU are combined, from a configuration viewpoint, the CPU and the SIMD ALU are connected to each other via a common data bus and a common address bus. Consequently, an operation is performed in such a manner that data is transferred from a memory to a register in the SIMD ALU and subjected to an arithmetic operation, a result of the arithmetic operation in the register is transferred to the memory, and then the next data process can be started. In this case, there is a problem such that an arithmetic efficiency cannot be raised by using data used by a neighboring processor element.

In a method devised to solve the problem, the SIMD ALU and a built-in memory are connected to each other via a local bus having a wide bus width independent of a system bus in accordance with the concept of a system LSI. According to the method, although the performance of transferring data between the SIMD ALU and the memory is improved, traffic of the system bus which does not limit an arithmetic instruction transferred from the CPU to the SIMD ALU becomes an issue, and an address generator is necessary for each of the CPU and the SIMD ALU, and the CPU cannot control both reading of data from the memory and storage of data in the SIMD ALU in a centralized manner.

Therefore, a problem such that the high-speed performance of the SIMD ALU cannot be effectively used arises.

DISCLOSURE OF INVENTION

A main object of the present invention is to realize a data processor capable of processing data at high speed.

Another object of the invention is to realize a data processor having an arithmetic and logic unit controlled by a central processing unit and connected to a memory via a local bus, wherein a central processing unit can control both reading of data from the memory and storage of data to the arithmetic and logic unit in a centralized manner.

Further another object of the invention is to realize a data processor capable of performing a high speed process of data by enabling an arithmetic operation to be executed every clock as much as possible by constantly supplying data to processor elements constructing an arithmetic unit.

To achieve the objects, a data processor of the invention is constructed by comprising an arithmetic and logic unit controlled by a CPU, first storage means, an address bus commonly connected to the CPU, the arithmetic and logic unit, and the first storage means, and a local data bus having a bus width wider than a data bus width of the CPU and connecting the arithmetic

and logic unit.

According to the invention, by providing the local data bus between the first storage means and the arithmetic and logic unit, the data transfer performance is improved. By connecting a control line from the CPU to the arithmetic and logic unit, an arithmetic instruction supplied to the arithmetic and logic unit is made independent of the traffic of the system bus. Further, since the address bus is commonly connected to the CPU, arithmetic and logic unit, and first storage means, it is sufficient to provide an address generator for only the CPU and it is unnecessary to provide the address generator for the arithmetic and logic unit. Together with the register of the arithmetic and logic unit, the first storage means is also in the address space of the CPU. Consequently, the CPU can control both reading of data from the first storage means and storage of data in the register of the arithmetic and logic unit in a centralized manner.

According to a preferred embodiment of the invention, the arithmetic and logic unit takes the form of an arithmetic and logic unit of an SIMD control type having a plurality of processor elements each having a first input terminal, a second input terminal, and an output terminal, and includes: a first register having a bit width equal to a total of bit widths of first input terminals of all of the processor elements; a

second register having a bit width equal to a total of bit widths of second input terminals of all of the processor elements; and a third register having a bit width equal to or wider than a bit width of the second input terminal of the processor element and capable of shifting data to the second register on a unit basis of the bit width of the second input terminal.

The data processor of the invention is, as will be described by the following embodiments, particularly effective on a motion estimating process or the like in an image encoding process. The invention can be applied to a processor which has to perform a high-speed arithmetic process in parallel with a process of the CPU.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram showing the configuration of a first embodiment of a data processor according to the invention.

FIG. 2 is a circuit diagram showing the internal configuration of an SIMD ALU 4 in FIG. 1.

FIG. 3 is a diagram showing the internal configuration of a CPU 2 in FIG. 1.

FIG. 4 is a diagram showing the internal configuration of a processor element 38 in FIG. 2.

FIG. 5 is a diagram for explaining the operation of the SIMD ALU 4 in FIG. 2.

FIG. 6 is a diagram for explaining the operation

of the SIMD ALU 4 in FIG. 2.

FIG. 7 is an explanatory diagram of reference image data used in the first embodiment.

FIG. 8 is an explanatory diagram of template image data used in the first embodiment.

FIG. 9 is an address map on a DRAM 16 in FIG. 1.

FIG. 10 is an address map on a work RAM 12 in FIG. 1.

FIG. 11 is an operation flowchart of the first embodiment.

FIG. 12 is a diagram for explaining the state of data transfer of a register in the SIMD ALU 4 in the first embodiment.

FIG. 13 is an explanatory diagram of an arithmetic range of a vector (0,0) in the first embodiment.

FIG. 14 is an explanatory diagram of an arithmetic range of a vector (1,0) in the first embodiment.

FIG. 15 is a block diagram showing the configuration of a second embodiment of a data processor according to the invention.

FIG. 16 is a diagram showing the internal configuration of a CPU in the second embodiment.

FIG. 17 is an operation flowchart of the second embodiment.

FIG. 18 is a block diagram showing the configuration of a third embodiment of a data processor

according to the invention.

FIG. 19 is a block diagram showing the configuration of a fourth embodiment of a data processor according to the invention.

FIG. 20 is a diagram showing the internal configuration of a VPU 160 in the fourth embodiment.

BEST MODE FOR CARRYING OUT THE INVENTION

First Embodiment

FIG. 1 is a block diagram showing the configuration of a first embodiment of a data processor according to the invention. The data processor of the embodiment performs a process of motion estimation according to a block matching method by an arithmetic and logic unit in an image encoding process. The configuration of the apparatus will be described first and the operation of the motion estimating process will be described later.

As shown in the diagram, the data processor has an arithmetic and logic unit 4 which takes the form of an SIMD ALU directly controlled by a central processing unit (hereinbelow, abbreviated as CPU) 2 via control lines 3 and 5, a work RAM 12 as storage means, an address bus 10 commonly connected to the CPU 2, the ALU 4 and the work RAM 12, and a local data bus 8 having a bus width wider than that of a data bus 6 of the CPU 2 and coupling the ALU 4 and the work RAM 12.

The CPU 2 decodes an instruction and controls the

whole. In the embodiment, an RISC type microprocessor is used. 20 denotes a ROM for storing a program of the CPU 2 and the like, 18 denotes a RAM for storing data, a program, or the like of the CPU 2, 12 indicates the work RAM for temporarily holding arithmetic data of the SIMD ALU 4, 16 indicates a DRAM in which image data is stored, 14 indicates a DRAM interface circuit between the DRAM 16 and the work RAM 12, and 22 expresses a DMA (Direct Memory Access) circuit for controlling a DMA transfer between the DRAM 16 and the work RAM 12.

The embodiment has three types of buses. The bus width of the data bus 6 of the CPU 2 is 32 bits, the bus width of the address bus 10 is 32 bits, and the bus width of each of the data buses 8 and 24 is 144 bits. In the drawing, each of the buses is added with an oblique line and the number indicative of the bus width (the number of bits).

The configuration and operation of each of the components will be described in detail hereinbelow.

FIG. 2 is a circuit diagram showing the internal configuration of the SIMD ALU 4 in Fig. 1. The ALU 4 takes the form of an SIMD control type arithmetic and logic unit having 16 processor elements 38, 40, ... 42, and 44 arranged in parallel. Each processor element has a first input terminal connected to a register 30 via a selector 32, a second input terminal connected to a register 34, and an output terminal connected to the data buses 6 and 8. The register 30 has a bit width

equal to a total of bit widths of the first input terminals of all the processor elements 38, 40, ... 42, and 44. The register 34 has a bit width equal to a total of bit widths of the second input terminals of all the processor elements. Further, a third resister 36 having a bit width wider than the bit widths of the second input terminals of the processor elements and capable of shifting data to the register 34 on the unit basis of the bit width of the second input terminal is also provided.

Each of the processor elements 38, 40, ... 42, and 44 is controlled by the CPU 2 via the control lines 3 and 5. The data supply from the register 30 to the processor elements 38, 40, ... 42, and 44 can be changed by the selector 32. In the registers 30, 34, and 36, data is written from write circuits 50, 46, and 48, respectively, controlled by the address bus 10 via the local bus 8.

FIG. 3 is a block diagram showing the configuration of the RISC type microprocessor 2 in Fig. 1. The configuration is quite similar to that of a conventionally known microprocessor, and includes an instruction decode circuit 58 for receiving and decoding an instruction fetched from an instruction fetch circuit 60, an ALU 64 for executing an instruction 68 from the instruction decode circuit 58, a program counter 54, and a general register 56.

Further, in the instruction decode circuit 58,

for example, in the case of an arithmetic instruction to the SIMD ALU 4, the signal line 3 is made active and, in the case of a read instruction of a result to the SIMD ALU 4, the signal line 5 is made active. 66, 68, 62, 73, and 74 denote instruction and data transfer lines.

FIG. 4 is a block diagram showing the configuration of the processor element. The 16 processor elements 38, 40, ... 42, and 44 of the SIMD ALU 4 have the same configuration. The processor element 38 will be described here as a representative example. The processor element 38 includes a register 82 for holding arithmetic results of ALUs 80 and 81, and a read control circuit 84 for controlling loading of data to the local data bus 8 or the data bus 6. To the ALU 80, nine bits as a part of the bit width of 144 bits of the register 30 are input via a bus 37 and nine bits as a part of the bit width of 144 bits of the register 34 are input via a bus 35. The input two data are subjected to arithmetic operation (subtraction) by the ALU 80, and an output of the ALU 80 is added with the value of the register 82 by the ALU 81. The arithmetic result of the ALU 81 is stored in the register 82.

FIGs. 5 and 6 are diagrams for explaining connection forms of the selector 32. In the first connection form, as shown in FIG. 5, nine bits a0 from the most significant bit out of 144 bits of the

register 30 are commonly supplied to the processor elements 38, 40, ... 44, and 42. In the second connection form, as shown in FIG. 6, all of 144 bits of the register 30 are supplied on the unit basis of nine bits from the most significant bit like a0, a2, ... a14, and a15 to the processor elements 38, 40, ... 44, and 42, respectively. Therefore, the data is distributed in such a manner that the nine-bit data of a0 shown in the diagram is supplied to the 0th processor element 38, the nine-bit data of a1 is supplied to the first processor element 40, and so on.

The case of performing the motion estimation of an image performed in a process of encoding an image signal according to the standard of the MPEG2 by using the data processor will now be described.

In the motion estimation of an image according to the standard MPEG2, a process of obtaining the position of a macro block on a reference screen to be compared, which is the most similar in a search range to a macro block to be encoded on the unit basis of a macro block having 16 pixels in the horizontal direction and 16 pixels in the vertical direction, and calculating a distance in an image frame between the two macro blocks. The motion estimation is usually performed by the block matching method. According to the block matching method, a process of accumulating a differential absolute value between a pixel in an image to be encoded and a pixel of a reference image with respect

to all the pixels of the macro block and finding the location of a macro block having the smallest accumulation value is performed.

FIG. 7 shows pixels of reference image data used to encode the image and FIG. 8 shows pixels of an encoded image as a macro block of an encoded image. It is assumed here that the reference image data has 352 pixels in the horizontal direction and 240 pixels in the vertical direction. Circled symbols ra1, ra2, ..., rb1, ... rp17, ... are symbols to identify pixels. The macro block has 16 pixels in the horizontal direction and 16 pixels in the vertical direction, and circled symbols ta1, ta2, ..., and tp16 are symbols to identify pixels.

FIG. 9 shows the state of data stored in the DRAM 16 of FIG. 1. Symbols in the diagram ra1, ra2, ... ta1, ... tb8, ... express pixels corresponding to the symbols shown in FIGs. 7 and 8. Addresses starting from A000 are assigned to areas of the reference image data, and four pixels in the horizontal direction are stored in 32 bits as the bit width of the DRAM 16. Addresses starting from B000 are assigned to macro blocks, that is, areas of encoded image data.

FIG. 10 shows encoded image data and reference image data stored in the work RAM 12. Addresses starting from C000 are assigned to areas of reference image data. Data of each pixel is nine-bit data. In 144 bits starting from the address C000, data of 16

pixels in the horizontal direction from the pixel ra1 to the pixel ra16 is stored. Pixels from the address D000 are assigned to areas of encoded image data. In a manner similar to the case of the reference image data, 16 pixels from the pixel ta1 to the pixel ta16 in the horizontal direction are stored in 144 bits of the address D000.

FIG. 11 is a process flowchart of the motion estimation in the data processor.

First, the data (FIG. 9) in the DRAM 16 is transferred to the work RAM 12 via the DRAM interface 14 (step 90). At this time, sign extension of adding a sign bit to eight-bit data per pixel to extend to nine-bit data per pixel is performed. By arranging data of four long words on the DRAM 16, data of 144 bits is created. Such a transfer is repeated and data is stored into the work RAM 12 via a bus 24.

Next, reference image data is transferred from the work RAM 12 to the register 34 of the SIMD ALU 4 via the local data bus 8 (step 92).

FIG. 12 is a diagram for explaining a detailed operation of step 92 and shows the relation between the flow of signals of the 16 processor elements 38, 40, ... 42, and 44 and the registers A 30, B 34, and C 36 of 144 bits and the time. Specifically, it also shows a change in the data of the registers 30, 34, and 36 with the time t in the vertical direction.

As described above, in the register A 30, plural

pixel data of an image to be encoded is stored. The upper nine bits in a series of bit string are commonly supplied to all the processor elements 38, 40, ... 42, and 44. Plural pixel data of a reference image is stored in the register B 34, and the data is supplied on the unit basis of nine bits to each of the processor elements in such a manner that the upper nine bits are supplied to the processor element 38, the following nine bits are supplied to the processor element 40, and so on. The register C 36 shifts data and supplies the shifted data to the register B34. In the case of an instruction of shifting nine bits, the upper nine bits of the register C 36 are supplied to the lower nine bits of the register B 34.

It is understood that at time t = 0 (step 92), the pixels from ra1 to ra16 of the reference image data in the register B 34 are transferred with the width of 144 bits at once.

At time t = 1 (step 94), data is transferred from the work RAM 12 to the register C 36. As a result, pixels from ra17 to ra32 of the reference image data are newly transferred at once to the register C 36 with the width of 144 bits. As a result, the reference image data of one line of 32 pixels in the horizontal direction is stored in both the registers B 34 and C 36.

At time t = 2 (step 96), data having the width of 144 bits from the macro block pixel ta1 to the pixel ta16 of the coded image data is transferred at once

from the work RAM 12 to the register A 30. All the data necessary for the arithmetic operation of the registers 30, 34, and 36 is stored.

At time t = 3 (step 98), simultaneous parallel arithmetic operation by the processor elements 38, 40, ... 42, and 44 and nine-bit shift of the registers 34 and 36 are performed. As a result, the processor element 38 executes an arithmetic operation of calculating a differential absolute value between the reference image data ra1 and the coded image data ta1. The result is stored in the register 82 in the processor element shown in FIG. 4. In the processor element 40, similarly, an arithmetic operation of calculating a differential absolute value between the reference image data ra2 and the coded image data ta1 is performed and the result is stored in the register 82 in the processor element 40. The arithmetic operation is similarly performed in the other processor elements 42, 44, and the like.

At time t = 4 (step 100), parallel arithmetic of a plurality of processor elements and shifting of nine bits by the registers 34 and 36 are performed again. As a result, in the processor element 38, an arithmetic operation of calculating the differential absolute value between the reference image data ra2 and the coded image data ta2 is performed. The resultant is added to the data of the register 82 and the resultant value is written in the register 82. In the processor

element 40, similarly, the arithmetic operation of obtaining the differential absolute value between the reference image data ra3 and the coded image data ta1 is executed, and the result is added to the value of the register 82 in the processor element.

The above operation is repeated, and the state of the register after performing the 16th arithmetic operation and shifting of nine bits by the registers 34 and 36 (step 102) is shown at time t = 18 in FIG. 12. When the range of the block matching is 16 pixels in the horizontal direction, the arithmetic operation of one horizontal line is finished at this time point.

In order to compute data of the immediately lower line, data is transferred from the work RAM 12 to the three registers 30, 34, and 36. First, at time t = 19 (step 104), data is transferred from the work RAM 12 to the register B.

At time t = 20 (step 106), data is transferred from the work RAM 12 to the register 36. As a result, the state at time t = 20 in FIG. 12 is obtained. The data of the reference image from pixel rb1 to pixel rb32 of one line below the calculated line is stored in the registers 34 and 36.

At time t = 21 (step 108), the data is transferred from the work RAM 12 to the register A. As a result, the pixels from ta1 to ta16 of the coded image of the calculated lower line are stored in the register A, and data is stored in all of the three

registers 30, 34, and 36. The arithmetic operation is executed in a manner similar to the above. Further, the operation is repeated for 16 lines.

As a result, an accumulated value of the differential values of all the pixels is stored in the register 82 in the processor element 38. The value expresses the result of the block matching of the vector (0, 0) in FIG. 13, that is, the degree of approximation to the vector (0, 0).

On the other hand, in the register 82 in the processor element 40, the result of the block matching computation of the vector (1, 0) in FIG. 14 is stored. Similarly, by the 16 processor elements 38 to 44, the results of the block matching arithmetic operation of 16 motion vectors can be calculated.

In the embodiment, a large amount of data can be transferred at once from the work RAM 12 to the SIMD ALU 4 not via the system data 8 of the data processor. The data transfer between the work RAM 12 and the SIMD ALU 4 can be controlled in a centralized manner by address management of the CPU 4 without providing the SIMD ALU 4 with an address generator. The invention is therefore effective on the data process requiring a number of arithmetic operations of the same type by a single instruction such as motion estimation of an image process performed by the block matching method.

Second Embodiment

FIG. 15 is a block diagram showing the configuration of a second embodiment of a data processor according to the invention. In the embodiment, a second SIMD ALU 130 is added to the data processor of FIG. 1. In association with this, control lines 134 and 132 from a CPU 131 are added. The internal configuration of the second SIMD ALU 130 is the same as that shown in FIG. 2, the same or corresponding elements are designated by the same numerals, and their description will not be given. The other elements substantially same as those in FIG. 1 are also designated by the same numerals and will not be described.

FIG. 16 is a block diagram showing the configuration of the CPU 131 in the second embodiment (FIG. 15). The configuration of the CPU 131 is substantially the same as that of the CPU 2 except for the point that the control lines 132 and 134 extended from an instruction decode circuit 133 are added to the CPU 2 in the first embodiment shown in FIG. 3. The control lines 132 and 134 are to control the second SIMD ALU 130.

FIG. 17 shows a processing flowchart for explaining the operation of the data processor of the second embodiment. In the second embodiment, the portion from the operation of storing data into three registers of the SIMD ALU 4, that is, the operation of transferring data from the DRAM 16 to the work RAM 12

(step 90) to the operation of transferring encoded image data from the work RAM 12 to the register A (step 96) is the same as that of the same step numbers in FIG. 11.

After the step 96, in the case of the second embodiment, data is registered in the register of the SIMD ALU 130. First, reference image data is transferred from the work RAM 12 to the register B (step 140). Next, the reference image data is transferred from the work RAM 12 to the register C (step 142). Finally, the encoded image data is transferred from the work RAM 12 to the register A (step 144). In a manner similar to the first embodiment, an arithmetic operation by the processor elements (PE) is executed. As a result, by simultaneously using 32 processor elements, block matching of different vectors can be performed, and the process can be carried out at higher speed.

Third Embodiment

FIG. 18 is a block diagram showing the configuration of a third embodiment of a data processor according to the invention. In the embodiment, two work RAMs 144 and 146 are provided, and the DRAM 16 side and the SIMD ALU 4 side are switched and used.

When data is stored in the work RAM 144 and the SIMD ALU 4 performs a signal process by using the data, the work RAM 144 is connected to the SIMD ALU 4 side by

the selectors 142 and 152. On the other hand, the work RAM 146 is connected to a DMAC 122 side by selectors 148 and 150. To the work RAM 146, the DMAC 122 transfers image data used next by the SIMD ALU 4 from the DRAM 16. After the SIMD ALU 4 finishes the signal process in the work RAM 144, a switch is made between the work RAMs 144 and 146. Specifically, the work RAM 144 is connected to the DMAC 122 side, and the work RAM 146 is connected to the SIMD ALU 4 side. With the configuration, data to be used is already transferred from the DRAM 16 to the work RAM 146, so that the SIMD ALU 4 can immediately start the arithmetic operation. Thus, the arithmetic efficiency can be increased.

Fourth Embodiment

FIG. 19 is a diagram showing a fourth embodiment of a data processor according to the invention. According to the embodiment, the data processor of the invention is provided in an image signal compression LSI.

Component blocks are connected to a bus 184 of a microprocessor unit 166. The component blocks include a communication interface 168 having the function of interface with an external modem, an audio interface 170 having the function of inputting/outputting an external audio signal, a video interface block 172 having the function of inputting/outputting an external video signal, an encoding/decoding block 164 for

encoding/decoding a variable length code, a Q-DCT/IQ-IDCT block 162 for performing quantization, inverse-quantization, DCT, and inverse-DCT, a DRAM control block 174 for controlling a DRAM 176, and a motion estimation block 160. The motion estimation block 160 is the same one described in the first embodiment.

The fourth embodiment is different from the apparatus shown in FIG. 1 with respect to the point that the DRAM 176 corresponding to the DRAM interface 14 and the DRAM 16 is provided on the outside of the LSI, and an MPU 166 has a control register 185 for controlling the motion estimation block 160. By the control register 185, the CPU 180 in the motion estimation block 160 is controlled.

The operation performed at the time of compressing an image in the configuration will be described. Encoded image data received by the video interface block 172 is once stored in the DRAM 176 and is loaded to the work RAM in the motion estimation block 160 on a macro block unit basis. At this time, reference image data of a corresponding search range is simultaneously loaded to the work RAM in the motion estimation block 160. As described in the first embodiment, differential absolute values of motion vectors are accumulated. After finishing the computation of all the vectors, a vector having the smallest differential absolute arithmetic value is set as a motion vector for the macro block. A differential

value between corresponding pixels of the coded image and the reference image at this time is calculated, and the result is sent to the Q-DCT/IQ-IDCT block 164. The Q-DCT/IQ-IDCT block 164 performs a DCT process and a quantizing process on the result sent from the motion estimation block 160, and transmits the result to the encode and decode block 164. The encode and decode block 164 executes a variable length encoding process, and the image data compressing process is finished.

As described above, by applying the invention to the image signal compression LSI, the high-performance image signal compression LSI having high programmability can be constructed.

INDUSTRIAL APPLICABILITY

As described by the foregoing embodiments, according to the invention, data can be constantly supplied to processor elements constructing an SIMD ALU and, particularly, an arithmetic efficiency in signal process of repeating an arithmetic process of a large data amount of compressing/decompressing an image signal can be raised.